ソースチェックに威力を発揮する Cプリプロセッサ

松井　潔
kmatsui@t3.rim.or.jp

目次

- □はじめに
- □私のプリプロセッサの概要
- □プリプロセス検証セットによる各種プリプロセッサの検証
- □プリプロセッサによるソースチェックはなぜ必要か
  - ○プリプロセッサによるソースチェックの影響力
  - ○glibc のソースを例に
- □cpp の実装方法
  - ○トークンベースの原則
  - ○関数型マクロの関数的展開
- □今後の update 計画
- □おわりに

私のプリプロセッサの概要

- きわめて正確である
- 詳細なテストをする検証セットが付属している
- 診断メッセージが豊富で親切である
- デバッグ情報を出力する #pragma ディレクティブを持っている
- Portable である
- オープンソースである
- 詳細なドキュメントが付属している

検証セットによる各種プリプロセッサの検証

検証セットの項目数と配点

	項目数	最低点	最高点
C90 規格合致度	173	-162	448
C99 規格合致度	19	0	96
C++98 規格合致度	8	0	24
品質:診断メッセージ	45	0	64
品質:その他	16	-40	111
計	261	-202	743

表 1: 検証セット V.1.3 の項目数と配点

検証セットによる各種プリプロセッサの検証

各種プリプロセッサの検証結果

ＯＳ	処理系	実行プログラム (版数)	(1)	(2)	注
OS-9/6x09	Microware C/6809	DECUS cpp	256	318	*1
MS-DOS		JRCPPCHK (V.1.00B)	400	449	*2
WIN32	Borland C++ V.4.02J	cpp32	412	458	*3
DJGPP V.1.12 M4	GNU C 2.7.1	cpp	452	556	*4
MS-DOS	LSI C-86 V.3.30c	cpp (改造版 beta13)	342	400	*5
FreeBSD, WIN32, etc.	GNU C, Borland C, etc.	cpp (V.2.0)	502	655	*6
WIN32	Borland C++ V.5.5	cpp32	412	465	*7
WIN32	LCC-Win32 V.3.6	lcc	392	479	*8
Linux, etc		ucpp (V.0.7)	423	491	*9
Linux, FreeBSD	GNU C 2.95.3	cpp	476	581	*10
Linux, FreeBSD, etc.	GNU C, LCC-Win32, etc.	cpp (V.2.3)	562	717	*11
					(1) 規格合致度 (2) 総合評価

表 2: 各種プリプロセッサの検証結果

追記

Linux	GNU C 3.2R	cpp	518	627

プリプロセッサによるソースチェックはなぜ必要か

- □プリプロセスの目的
  - ○readability, メンテナンス性, portability の改善
- □問題のソースが多く存在する背景
  - ○C90 以前のプリプロセス仕様のあいまいさ
  - ○寡黙すぎるプリプロセッサ
- □プリプロセッサによるソースチェックの影響力
  - ○FreeBSD 2.2.2 の kernel ソースと libc ソース
- □glibc のソースを例に

glibc のソースを例に

- 行をまたぐ文字列リテラル
- プリプロセスを要する *.S ファイル
- ’defined’ に展開されるマクロ
- 関数型マクロとして展開されるオブジェクト型マクロ
- undocumented な環境変数の仕様

行をまたぐ文字列リテラル

```
#define ELF_MACHINE_RUNTIME_TRAMPOLINE asm ("
    .text
    .globl _dl_runtime_resolve
    etc. ...
");
```

’defined’ に展開されるマクロ

```
#define HAVE_MREMAP defined(__linux__) && !defined(__arm__)

#if HAVE_MREMAP


defined(__linux__) && !defined(__arm__)              (1)

defined(1) && !defined(__arm__)                    (2)


#if defined(__linux__) && !defined(__arm__)
#define HAVE_MREMAP 1
#endif
```

関数型マクロとして展開されるオブジェクト型マクロ

```
#define CHAR_CLASS_TRANS SWAPU16

#define SWAPU16(w) ((((w) >> 8) & 0xff) | (((w) & 0xff) << 8))


#define CHAR_CLASS_TRANS(w) SWAPU16(w)
```

cpp の実装方法

- □目標としたこと
  - ○豊富で的確な診断メッセージ
  - ○全仕様をドキュメントに記載
  - ○多くの処理系に実装
  - ○網羅的な検証セットの作製
- □トークンベースの原則
  - ○文字ベースの処理を紛れ込ませない
  - ○マクロ展開によって生成されたトークンもチェック
- □関数型マクロの関数的展開
  - ○引数付きマクロの混乱の歴史
  - ○function-like の原則の徹底

今後の update 計画

- 各処理系付属のプリプロセッサとの互換性の向上
- 対応する処理系を増やす
  - Microsoft Visual C++ 等
- GNU C 3.2 / cpp に対応させる
  - test-suite の開発にコミットしたい
- 英語版ドキュメントを作成する

CVS リポジトリをオープン

- □開発中の最新版を cvs リポジトリに
  - ○cvs@cvs.m17n.org:/cvs/matsui-cpp
- □ダウンロードするには
  - ○http://cvs.m17n.org から入って matsui-cpp に
- □開発に参加するには
  - ○リポジトリの説明は http://cvs.m17n.org から
  - ○ssh の公開カギをメールする
  - ○kmatsui@t3.rim.or.jp または cvs@cvs.m17n.org に